# Tools

Node.js
npm
VS Code
Svelte für VS Code (James Birtles)

# Repo

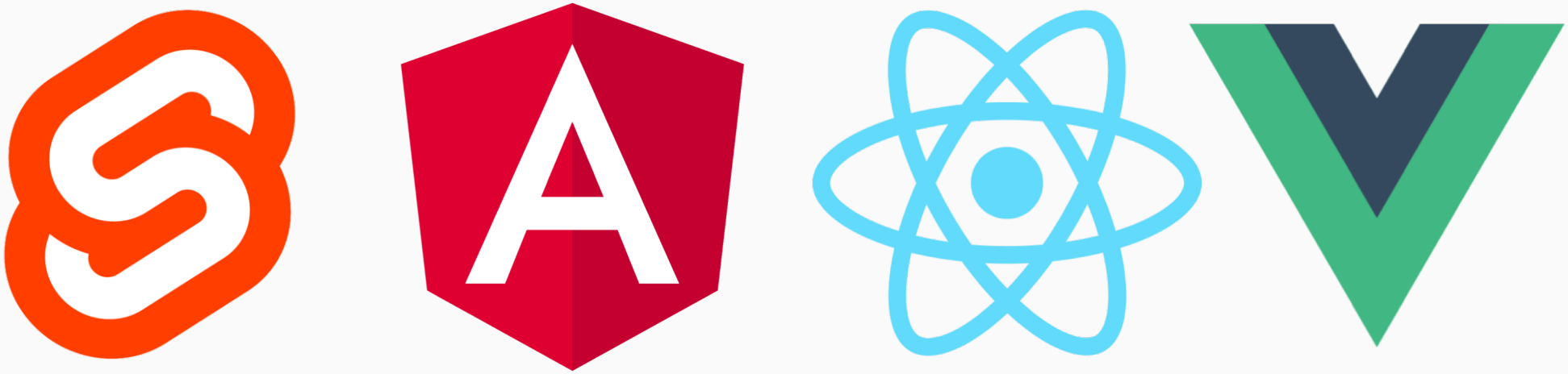
https://github.com/lukaskorten/**svelte-workshop**

LUKAS KORTEN

adesso AG


SVELTE

# Ein Framework?



# Ein ~~Framework~~ Tool

Anwendungs-
code

Buildprozess

JS HTML CSS

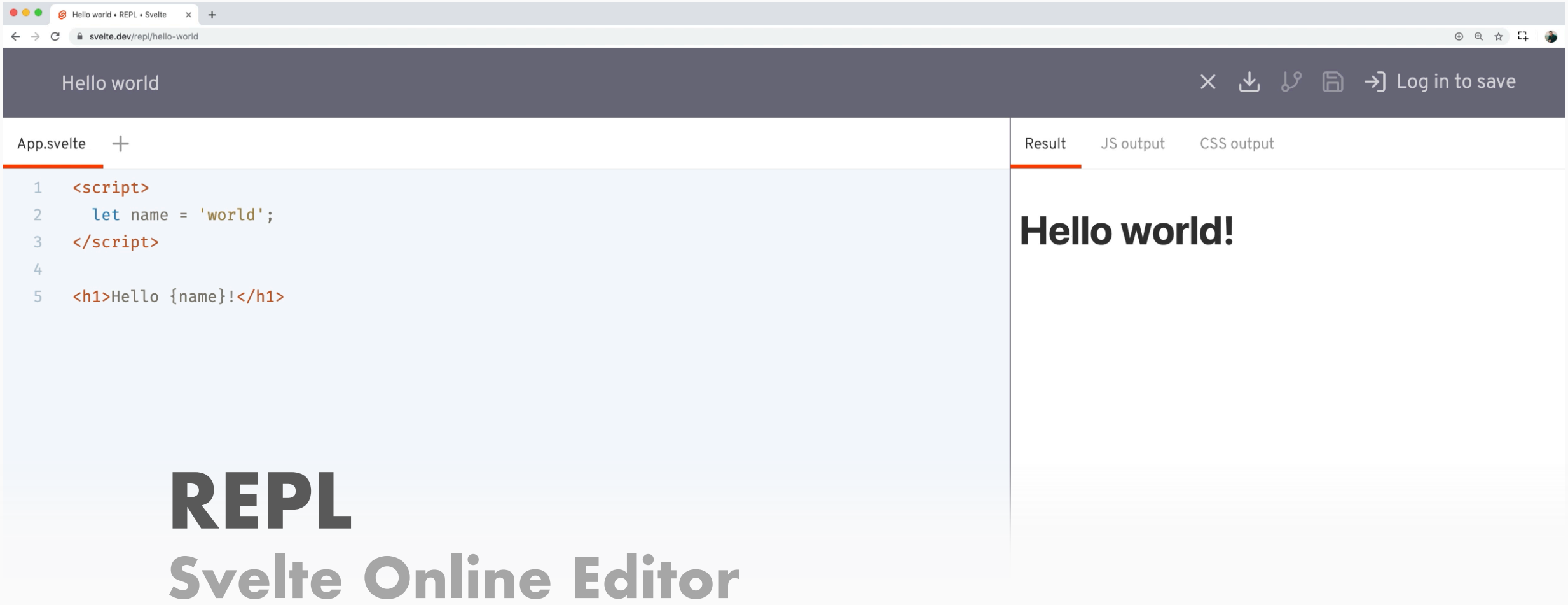
BROWSER

- **kein** virtueller DOM-Vergleich
- sehr **effizient** und **sparsam**
- unterstützt **Barrierefreiheit**

- **Konzepte vergleichbar** mit React, Angular und Vue
- Produktionscode ist **klein** und **schnell**
- **eine Alternative** für kleine und schnelle Apps

# Projekt anlegen

zwei Möglichkeiten um loszulegen




# REPL

## Svelte Online Editor

# degit

```
npx degit sveltej
cd my-svelte-proj
npm install
npm run dev
```

# Component

die Struktur einer Svelte-Component

## Eine einfache Svelte Component

```
<h1>Hallo Welt!</h1>
```

## Struktur einer Svelte Component

```
<script>
  // JavaScript-Logik
</script>

<style>
  /* Styling */
</style>

<!-- Markup -->
<h1>Hallo Welt!</h1>
```

# Template Syntax

Daten mit Markup verknüpfen

## Variablen im Markup referenzieren

```
<script>
  let name = "Welt";
</script>

<h1>Hallo {name}!</h1>
```

## JavaScript-Ausdrücke im Markup

```
<script>
  let name = "Welt";
</script>

<h1>Hallo {name.toUpperCase()}!</h1>
```

## Dynamisches
## Setzen von Attributen

```
<script>
  let src = "images/image.gif";
</script>

<img />
```

## Dynamisches
## Setzen von Attributen

```
<script>
  let src = "images/image.gif";
</script>

<img src={src}/>
```

## Kurzschreibweise für gleichnamige Variablen und Attribute

```
<script>
  let src = "images/image.gif";
</script>

<img {src} />
```

## JavaScript-Ausdrücke in Strings

```
<script>
  let src = "images/image.gif";
  let name = "Max";
</script>

<img {src} alt="{name} lernt Svelte." />
```

## Mit **on:** auf DOM-Events lauschen

```
<script>
  let name = "Luke";
</script>

<input type="text" value={name} />
<p>Hallo {name}!</p>
```

## Mit **on:** auf DOM-Events lauschen

```
<script>
  let name = "Luke";
</script>

<input type="text" value={name} on:input={updateName} />
<p>Hallo {name}!</p>
```

## Auf Events reagieren

```
<script>
  let name = "Luke";
  function updateName(event) {
    name = event.target.value;
  }
</script>

<input type="text" value={name} on:input={updateName} />
<p>Hallo {name}!</p>
```

## Bidirektionale Bindung

```
<script>
  let name = "Luke";
</script>

<input type="text" bind:value={name} />
<p>Hallo {name}!</p>
```

# Reactivity

State und DOM synchron halten

## Zuweisungen - Das DOM auf dem neuesten Stand halten

```
<script>
  let count = 0;
</script>

<button>
    {count} Mal geklickt
</button>
```

## Zuweisungen - Das DOM auf dem neuesten Stand halten

```
<script>
  let count = 0;

  function handleClick() {
    // Event-Handler
  }
</script>

<button>
  {count} Mal geklickt
</button>
```

## Zuweisungen - Das DOM auf dem neuesten Stand halten

```
<script>
  let count = 0;

  function handleClick() {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  {count} Mal geklickt
</button>
```

## Reaktive Deklarationen

```
let count = 0;
$: doubled = count * 2;
```

## Reaktive Deklarationen

```
let count = 0;
$: doubled = count * 2;
```

## Reaktive Deklarationen

```
let count = 0;
$: doubled = count * 2;
```

## Reaktive Deklarationen

```
<script>
  let count = 0;
  $: doubled = count * 2;

  function handleClick() {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  {count} Mal geklickt
</button>
<p>{count} mal 2 ist {doubled}</p>
```

## Reaktive Anweisungen

```
let count = 0;

$: console.log(`Die Klickzahl ist ${count}`);
```

## Reaktive Anweisungen als Block

```
let count = 0;

$: {
    console.log(`Die Klickzahl ist ${count}`);
    alert(`Hier kommt die Klickzahl ${count}`);
}
```

## Reaktive Anweisung mit einem if-Block

```
let count = 0;

$: if (count > 10) {
    console.log("Die Klickzahl ist zu hoch!");
}
```

## Arrays und Objekte aktualisieren

```
<script>
  let numbers = [1, 2, 3];
  $: sum = numbers.reduce((sum, value) => sum + value, 0);

  function addNumber() {
    // Neue Zahl hinzufügen
  }
</script>


<button on:click={addNumber}>Eine Zahl hinzufügen</button>
<p>{numbers.join(' + ')} = {sum}</p>
```

# Arrays und Objekte aktualisieren

```
<script>
  let numbers = [1, 2, 3];
  $: sum = numbers.reduce((sum, value) => sum + value, 0);

  function addNumber() {
    numbers.push(numbers.length + 1);
  }
</script>


<button on:click={addNumber}>Eine Zahl hinzufügen</button>
<p>{numbers.join(' + ')} = {sum}</p>
```

# Arrays und Objekte aktualisieren

```
<script>
  let numbers = [1, 2, 3];
  $: sum = numbers.reduce((sum, value) => sum + value, 0);

  function addNumber() {
    numbers.push(numbers.length + 1);
    numbers = numbers;
  }
</script>

<button on:click={addNumber}>Eine Zahl hinzufügen</button>
<p>{numbers.join(' + ')} = {sum}</p>
```

## Arrays und Objekte aktualisieren

```
<script>
  let numbers = [1, 2, 3];
  $: sum = numbers.reduce((sum, value) => sum + value, 0);

  function addNumber() {
    numbers = [...numbers, numbers.length + 1];
  }
</script>


<button on:click={addNumber}>Eine Zahl hinzufügen</button>
<p>{numbers.join(' + ')} = {sum}</p>
```

# if

bedingte Ausdrücke

# If-Block

```
<script>
  let showAlert = false;
</script>

<p>Eine sehr wichtige Meldung!</p>
```

# If-Block

```
<script>
  let showAlert = false;
</script>

{#if showAlert}
  <p>Eine sehr wichtige Meldung!</p>
{/if}
```

# If-Block

```
<script>
  let showAlert = false;


  function toggleAlert() {
    showAlert = !showAlert;
  }
</script>

<button on:click={toggleAlert}>Klick mich</button>

{#if showAlert}
  <p>Eine sehr wichtige Meldung!</p>
{/if}
```

# If-Block

```
<script>
  let showAlert = false;
  $: label = showAlert ? "Ausblenden" : "Einblenden";

  function toggleAlert() {
    showAlert = !showAlert;
  }
</script>

<button on:click={toggleAlert}>{label}</button>

{#if showAlert}
  <p>Eine sehr wichtige Meldung!</p>
{/if}
```

## If-Block

```
...

<button on:click={toggleAlert}>{label}</button>

{#if showAlert}
  <p>Eine sehr wichtige Meldung!</p>
{/if}
```

## If-Else-Block

```
...

<button on:click={toggleAlert}>{label}</button>

{#if showAlert}
  <p>Eine sehr wichtige Meldung!</p>
{:else}
  <small>Es gibt nichts zu berichten ...</small>
{/if}
```

# each

Schleifen im Markup

## each-Block

```
<script>
  let frameworks = ["Svelte", "Angular", "React", "Vue"];
</script>

<ul>
  {#each frameworks as framework}
    <li>{framework}</li>
  {/each}
</ul>
```

## each-Block

```
<script>
  let frameworks = ["Svelte", "Angular", "React", "Vue"];
</script>

<ul>
  {#each frameworks as framework, i}
    <li>{i + 1}: {framework}</li>
  {/each}
</ul>
```

# Styling

CSS-Kapselung

## Styling

```
<p>Ich bin ein Paragraf</p>
```

```
<p class="svelte-85y15e">Ich bin ein Paragraf</p>
```

```
  }
</style>

<p>Ich bin ein Paragraf</p>
```

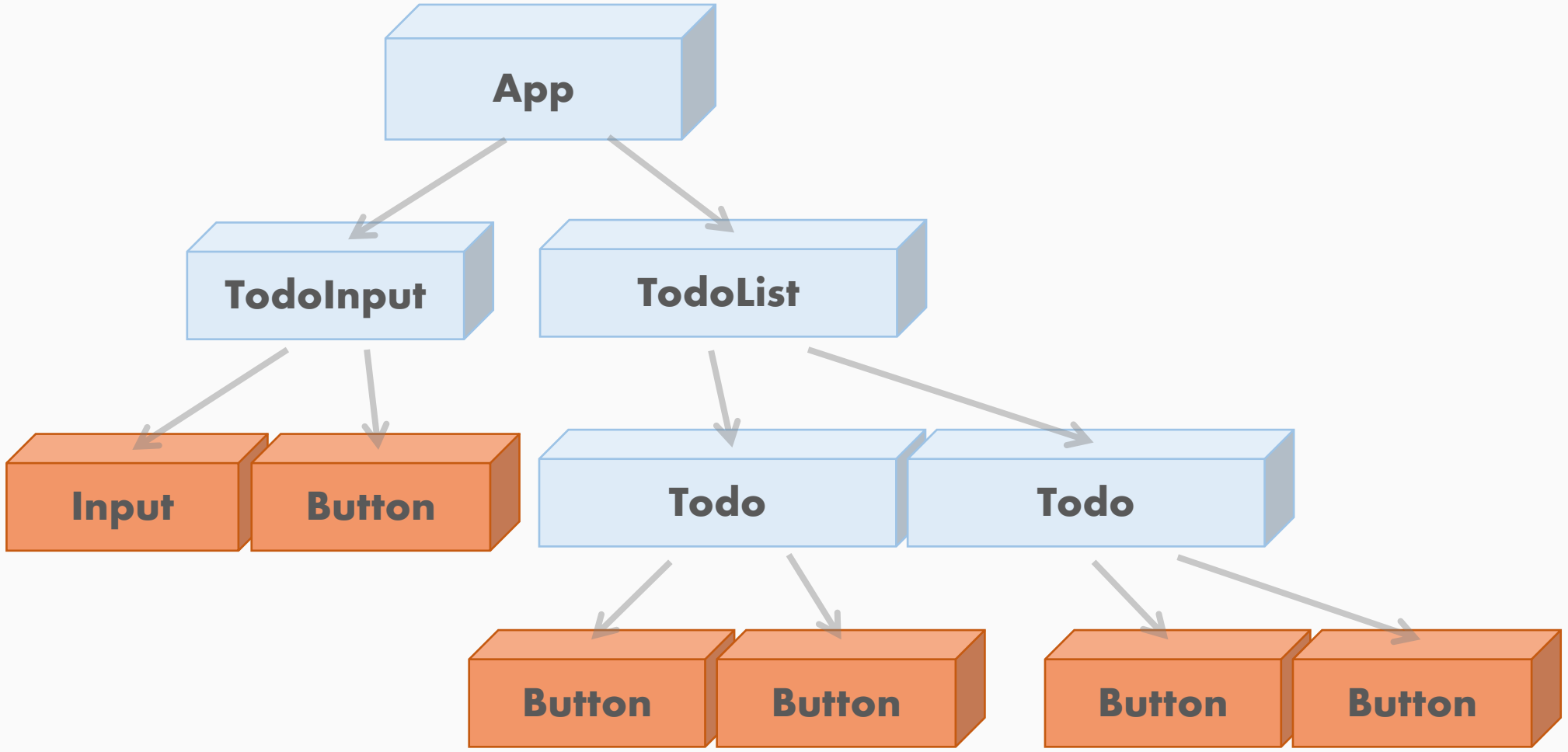
# Übung 1

Template-Syntax

# Nested Components

Components importieren

# Verschachtelte Komponenten



## Verschachtelte Komponenten

```
<style>
  p {
    color: turquoise;
    font-family: "Courier New";
    font-size: 1.5em;
  }
</style>


<p>Ich bin ein Paragraf</p>
```

## Verschachtelte Komponenten

```
<p>Ich bin ein weiterer Paragraf</p>
```

## Verschachtelte Komponenten

```
<style>
  p {
    color: turquoise;
    font-family: "Courier New";
    font-size: 1.5em;
  }
</style>


<p>Ich bin ein Paragraf</p>
```

## Verschachtelte Komponenten

```
<script>
  import Nested from "./Nested.svelte";
</script>

<style>
  p { ... }
</style>

<p>Ich bin ein Paragraf</p>

<Nested />
```

# Properties

der Input einer Component

## Properties

```
<script>
  let name = "Max";
</script>

<p>Ich bin {name}!</p>
```

## Properties

```
<script>
  export let name;
</script>

<p>Ich bin {name}!</p>
```

## Properties

```
<script>
  export let name = "Max";
</script>

<p>Ich bin {name}!</p>
```

## Properties

```
<script>
  import Child from "./Child.svelte";
  let childsName = "Jessica";
</script>

<Child name={childsName} />
```

# Component Events

benutzerdefinierte Events

## Benutzerdefinierte Events

### Greeter.svelte

```
<button on:click={greet}>Sag Hallo</button>
```

## Benutzerdefinierte Events

**Greeter.svelte**

```
<script>
  export let name;
  function greet() {
    // Event erzeugen
  }
</script>

<button on:click={greet}>Sag Hallo</button>
```

## Benutzerdefinierte Events

**Greeter.svelte**

```
<script>
  import { createEventDispatcher } from "svelte";

  const dispatch = createEventDispatcher();

  export let name;
  function greet() {
     // Event erzeugen
  }
</script>

<button on:click={greet}>Sag Hallo</button>
```

## Benutzerdefinierte Events

**Greeter.svelte**

```
<script>
  import { createEventDispatcher } from "svelte";

  const dispatch = createEventDispatcher();

  export let name;
  function greet() {
    dispatch("greet", `Hallo ${name}!`);
  }
</script>

<button on:click={greet}>Sag Hallo</button>
```

## Benutzerdefinierte Events

**App.svelte**

```
<script>
  import Greeter from "./Greeter.svelte";
</script>

<Greeter name="Max" />
```

## Benutzerdefinierte Events

**App.svelte**

```
<script>
  import Greeter from "./Greeter.svelte";
</script>

<Greeter name="Max" on:greet={showGreeting} />
```

## Benutzerdefinierte Events

**App.svelte**

```
<script>
  import Greeter from "./Greeter.svelte";

  function showGreeting(event) {
    alert(event.detail);
  }
</script>

<Greeter name="Max" on:greet={showGreeting} />
```

# each & keys

eindeutige Schlüssel für Listenelemente

## Each-Block

```
<script>
  let frameworks = [
    { id: 1, name: "Angular" },
    { id: 2, name: "React" },
    { id: 3, name: "Vue" },
    { id: 4, name: "Svelte" }
  ];
</script>

<ul>
  {#each frameworks as framework}
    <li>{framework.name}</li>
  {/each}
</ul>
```

## Each-Block

```
<script>
  let frameworks = [
    { id: 1, name: "Angular" },
    { id: 2, name: "React" },
    { id: 3, name: "Vue" },
    { id: 4, name: "Svelte" }
  ];
</script>

<ul>
  {#each frameworks as framework, i}
    <li># {i} {framework.name}</li>
  {/each}
</ul>
```

## Each-Block Schlüssel

**Framework.svelte**

```
<script>
  export let name;
  export let id;
  let state = `state: ${name}, ${id}`;
</script>

<p>{name} ({id}) – {state}</p>
```

## Each-Block Schlüssel

**App.svelte**

```
<script>
  import Framework from "./Framework.svelte";
  let frameworks = [
    { id: 1, name: "Angular" },
    { id: 2, name: "React" },
    { id: 3, name: "Vue" },
    { id: 4, name: "Svelte" }
  ];
  function removeFirst() {
    frameworks = frameworks.slice(1);
  }
</script>

...
```

## Each-Block Schlüssel

**App.svelte**

```
<script> ... </script>

<button on:click={removeFirst}>Erstes Element entfernen</button>

{#each frameworks as framework}
  <Framework name={frameworks.name} id={frameworks.id} />
{/each}
```

## Each-Block Schlüssel

**App.svelte**

```
<script> ... </script>

<button on:click={removeFirst}>Erstes Element entfernen</button>

{#each frameworks as framework (framework.id)}
  <Framework name={frameworks.name} id={frameworks.id} />
{/each}
```

## Each-Block Schlüssel

**App.svelte**

```
<script> ... </script>

<button on:click={removeFirst}>Erstes Element entfernen</button>

{#each frameworks as framework (framework.id)}
  <Framework {...framework} />
{/each}
```

# Übung 2

Nested Components

# Events

… und ihre Modifier

## Events

```
on:eventname={handler}
```

## Events

```
<script>
  function clickHandler(event) {

    console.log("Ich wurde geklickt");
  }
</script>

<button on:click={clickHandler}>
  Klick mich!
</button>
```

## Event
## Modifier

```
<script>
  function clickHandler(event) {
    event.preventDefault();
    console.log("Ich wurde geklickt");
  }
</script>

<button on:click={clickHandler}>
  Klick mich!
</button>
```

## Event
## Modifier

```
<script>
  function clickHandler() {
    console.log("Ich wurde geklickt");
  }
</script>

<button on:click|preventDefault={clickHandler}>
  Klick mich!
</button>
```

## Event
## Modifier

```
<form on:submit|preventDefault={handleSubmit}>
  <!-- Verhindert das Neuladen der Seite -->
</form>
```

## Events

```
on:eventname|modifier={handler}
```

## Event
## Modifier

**preventDefault** – *ruft* `event.preventDefault()` *vor Ausführung des Event-Handlers auf*

**stopPropagation** – *ruft* `event.stopPropagation()` *auf und unterbricht die weitere Verbreitung*

## Event
## Modifier

**passive** – *verbessert Scroll-Performanz bei `touch/wheel`-Events*

**capture** – *ruft den Event-Handler in der `capture`-Phase anstatt in der `bubbling`-Phase auf*

**once** – *entfernt den Event-Handler nach dem ersten Aufruf*

## Events weiterleiten

```
on:eventname={handler}
```

## Events weiterleiten

**on:***eventname*

## Events weiterleiten

```
...

<button on:click={handler}>
  <slot/>
</button>
```

## Events weiterleiten

```
...

<button on:click >
  <slot/>
</button>
```

## Events weiterleiten

```
<FancyButton on:click={handler}>
  Klick mich!
</FancyButton>
```

# Slots

Platzhalter für
benutzerdefiniertes Markup

## Slots
## Alert.svelte

```
<style>
  .alert {
    padding: 1rem;
    width: 400px;
    background: white;
    border-radius: 6px;
    border: 1px solid #eeeeee;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
  }
</style>

<div class="alert">

</div>
```

## Slots
## Alert.svelte

```
<style>
  .alert {
    padding: 1rem;
    width: 400px;
    background: white;
    border-radius: 6px;
    border: 1px solid #eeeeee;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
  }
</style>

<div class="alert">
  <slot />
</div>
```

## Slots
### Alert.svelte

```
<script>
  import Alert from "./Alert.svelte";
</script>

<Alert>
  <h1>Warnung</h1>
  <p>Ihre Benutzerdaten konnten nicht gespeichert werden</p>
</Alert>
```

# Lifecycle

Callbacks im Lebenszyklus
einer Component

Lifecycle

# onMount

```
onMount(callback: () => void)
```

Lifecycle

# onMount

```
<script>
  import { onMount } from "svelte";

  onMount(() => {
    console.log("wurde in das DOM eingebunden");
  });
</script>
```

Lifecycle

# onMount

```
onMount(callback: () => () => void)
```

Lifecycle

# onMount

```
<script>
  import { onMount } from 'svelte';

  onMount(() => {
    const interval = setInterval(() => {
      console.log('beep');
    }, 1000);

    return () => clearInterval(interval);
  });
</script>
```

Lifecycle

# onDestroy

```
onDestroy(callback: () => void)
```

Lifecycle

# onDestroy

```
<script>
  import { onDestroy } from "svelte";

  const interval = setInterval(() => {
    console.log("beep");
  }, 1000);

  onDestroy(() => {
    clearInterval(interval)
  });
</script>
```

Lifecycle

# beforeUpdate & afterUpdate

```
beforeUpdate(callback: () => void)
afterUpdate(callback: () => void)
```

Lifecycle

# beforeUpdate & afterUpdate

```
<script>
  import { beforeUpdate, afterUpdate } from 'svelte';

  beforeUpdate(() => {
    console.log('vor Aktualisierung');
  });

  afterUpdate(() => {
    console.log('nach Aktualisierung');
  });
</script>
```

Lifecycle

# tick

```
promise: Promise = tick()
```

Lifecycle

# tick

```
<script>
  import { beforeUpdate, tick } from 'svelte';

  beforeUpdate(async () => {
    console.log('die Aktualisierung steht aus');
    await tick();
    console.log('gerade aktualisiert');
  });
</script>
```

# Übung 3

Bindings, Slots und Event-Modifier